Application Note: JN-AN-1069

IEEE 802.15.4 Serial Cable Replacement

This Application Note describes how to create a wireless UART link between the UARTs of two NXP JN516x or JN517x wireless microcontrollers on modules fitted to carrier boards. The features of the application include:

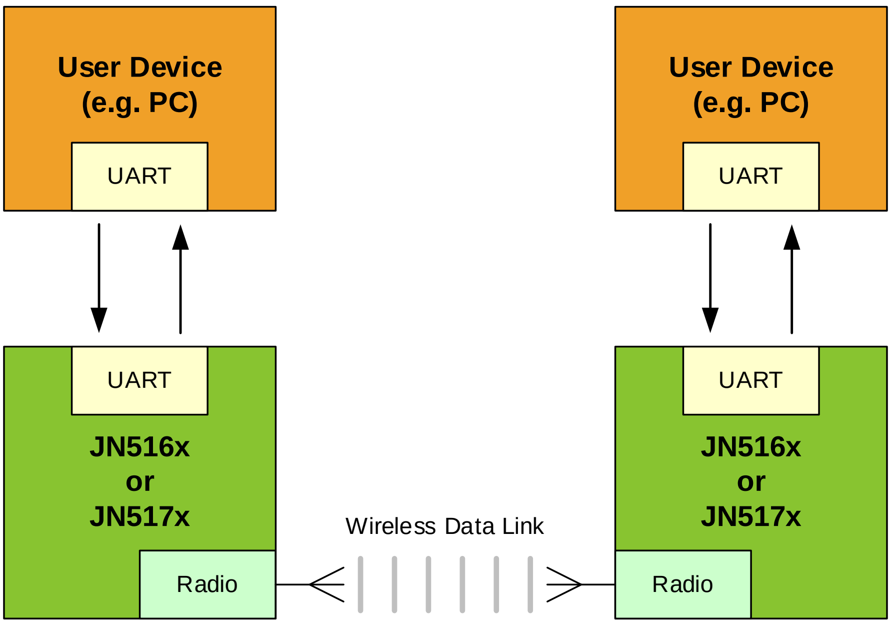
- **Replacement of serial cables with wireless connectivity**
- **Quick addition of low-cost wireless connectivity to products using a UART link for communications**
- **UART flow control using hardware with the RTS/CTS lines or software with the XON/XOFF protocol to control the flow of data into and out of the UART**
- **Radio flow control using a protocol to control the flow of data over the radio**

This application was developed using APIs from the NXP IEEE 802.15.4 Software Developer's Kit (SDK) [JN-SW-4163 for JN516x, JN-SW-4263 for JN517x], available free-of-charge via the Wireless Connectivity area of the NXP web site.

1 Application Overview

The application uses a pair of NXP JN516x or JN517x modules fitted to carrier boards (DR1174 for JN516x, OM15028 for JN517x) to form a wireless network. The required hardware is available in NXP kits, including JN516x-EK001, JN516x-EK004 and JN517x-DK005.

Data received by a UART on one JN516x/7x module is transmitted via the wireless radio link to the second JN516x/7x module, where it is output by the second module's UART, and vice versa. This allows two devices (such as PCs) to communicate via a wireless radio link. This is illustrated in the figure below:



Two applications, Coordinator and End Device, are provided as part of this demonstration:

1. The Coordinator initially creates the network, then runs in the same way as the End Device.
2. The End Device initially joins the network, then runs in the same way as the Coordinator.

It is assumed that a permanent power source is available at both ends of the wireless link, allowing the radio to always be active and ready to transmit or receive data.

2 Running the Demonstration

This section describes how to use the supplied pre-built binaries to run the demonstration.

2.1 Loading the Applications

The table below lists the application binary files supplied with this Application Note and indicates the hardware components with which the binaries can be used. These files are located in the **Build** directories for the relevant applications.

Application	**JN5168 Binary File**	**Hardware (e.g. from JN516x-EK001)**
Coordinator	**Coordinator_JN5168.bin**	DR1174 Carrier Board with JN5168 module DR1215 LCD Expansion Board (optional) *
End Device	**EndDevice_JN5168.bin**	DR1174 Carrier Board with JN5168 module DR1215 LCD Expansion Board (optional) *
Application	**JN5169 Binary File**	**Hardware (e.g. from JN516x-EK004)**
Coordinator	**Coordinator_JN5169.bin**	DR1174 Carrier Board with JN5169 module
End Device	**EndDevice_JN5169.bin**	DR1174 Carrier Board with JN5169 module
Application	**JN5179 Binary File**	**Hardware (e.g. from JN517x-DK005)**
Coordinator	**Coordinator_JN5179.bin**	OM15028 Carrier Board with JN5179 module
End Device	**EndDevice_JN5179.bin**	OM15028 Carrier Board with JN5179 module

Table 1: Application Binaries and Hardware Components

* For information on using the LCD Expansion Board, refer to Section 2.4.

A binary file can be loaded into the Flash memory of a JN516x/7x device using the JN51xx Flash Programmer (JN-SW-4107), available via the NXP web site. This software tool is described in the *JN51xx Production Flash Programmer User Guide (JN-UG-3099)*.

Note: You can alternatively load a binary file into a JN516x/7x module using the Flash programmer built into the relevant IDE (see Section 5).

To load an application binary file into a JN516x/7x module, follow the instructions below:

1. Connect a USB port of your PC to the USB Mini B port on the carrier board using a ‘USB A to Mini B’ cable. At this point, you may be prompted to install the driver for the cable.
2. Determine which serial communications port on your PC has been allocated to the USB connection.
3. On your PC, open a command window.
4. In the command window, navigate to the Flash Programmer directory:

C:\NXP\ProductionFlashProgrammer

5. Run the Flash programmer to download your binary file to JN516x/7x Flash memory by entering a command with the following format at the command prompt:

```
JN51xxProgrammer.exe –s <comport> -f <path to .bin file>
```

where `<comport>` is the number of the serial communications port.

6. Once the download has successfully completed, disconnect the USB cable and, if required, reset the board or module to run the application.

2.2 Starting the Applications

Connect each carrier board to a PC using a 'USB A to Mini B' cable. UART0 on the JN516x/7x device is used by default.

- When the boards are first powered on, LEDs 0 and 1 (marked as D6 and D3 on the DR1174 board, and as D3 and D2 on the OM15028 board) flash alternately while the node is creating or joining the network.
- Once the node is a member of a network, LEDs 0 and 1 flash together while the node attempts to pair with another node in the network.
- When the node has paired, LED 0 is illuminated while transmit is enabled on the UART and LED 1 is illuminated while receive is enabled on the UART.

2.3 Using the Applications

> **!** ***Caution:*** *Ensure the JN51xx Flash Programmer software is disconnected from the serial port on the PC, if using the same serial port for programming and running the application.*

A terminal emulator can be used to send data between the nodes. The serial connection is 115200 bps, 8 data bits, no parity, and 1 stop bit with hardware flow control, by default.

Data entered into one node's terminal emulator is passed to the node's UART and then transmitted over the radio to the other node. Here, it is received and passed via the node's UART to that node's terminal emulator.

Whenever data is transmitted, LED 0 is extinguished while the node waits for an acknowledgement from its paired node.

2.4 Using the LCD Expansion Board

When the Coordinator or End Device binary file is loaded into a JN516x/7x module on a carrier board that is also fitted with an LCD Expansion Board (DR1215), the LCD screen will display a looping count of the data bytes transmitted and received by the various modules of the application, along with the rate of throughput (bytes per second) of the queues.

Only the JN516x-EK001 evaluation kit (containing JN5168 modules) includes the LCD Expansion Board. Therefore, the option to display this information on the LCD screen is enabled by default only for JN5168 applications. You can use the LCD Expansion Board with the hardware from the JN516x-EK004 and JN517x-DK005 kits but the LCD display option must be enabled for a JN5169 or JN517x application – in the header file **wuart.h**, the define WUART_LCD for the relevant chip type must be changed from FALSE to TRUE. You will then need to rebuild the application as described in Section 6.

3 Software Design

This Application Note was created using the C function APIs (Application Programming Interfaces) of the NXP IEEE 802.15.4 SDK (JN-SW-4163 for JN516x, JN-SW-4263 for JN517x), which contain software libraries and tools for developing wireless networking applications to run on the JN516x/7x wireless microcontrollers.

The main APIs used were as follows:

- The IEEE 802.15.4 APIs provide functions for controlling the wireless network.
- The JN516x or JN517x Integrated Peripherals API provides functions for controlling the on-chip hardware peripherals of the JN516x/7x devices.

A User Guide (JN-UG-3024) and other detailed information about IEEE 802.15.4 can be found in the Wireless Connectivity area of the NXP web site.

Apart from the creation and joining of the network, the software contained in each node is identical.

3.1 Overview of Software Modules

The source files required to build the application are described below:

3.1.1 config.h

This file contains definitions controlling the operation of the radio network, including the channels to be used to form the network and the PAN ID of the network to be formed.

3.1.2 crd_coordinator.c/h

The Coordinator node is responsible for creating the network and allowing other nodes to join the network as children of the Coordinator.

crd_coordinator.c contains the standard set of functions for starting and running an IEEE 802.15.4 Coordinator node.

3.1.3 ed_enddevice.c/h

An End Device node can join the network created by the Coordinator node.

ed_enddevice.c contains the standard set of functions for starting and running an IEEE 802.15.4 End Device node.

3.1.4 node.c/h

node.c provides networking code common to both the Coordinator and End Device nodes.

Functions are provided to initialise the stack and application, maintain a timer for the application and transmit data via the network.

The data used by this module is held in the global **NODE_sData** structure defined in **node.h**.

3.1.5 wuart.c/h

This module provides the main functionality of this wireless UART application.

The module runs a state machine when the node becomes a member of the network. This state machine initially acts to form a pairing with another unpaired node in the network. Once paired, the state machine manages the exchange of data between the paired nodes forming a wireless UART. Further detail on this state machine is provided in the Wireless UART State Machine section of this document.

Two circular queues (implemented by **queue.c**) are allocated:

1. The receive queue is filled with data received by the UART (in **uart.c**). This module takes the data from the receive queue and transmits it over the radio to the paired node.
2. The transmit queue is filled with data received over the radio by this module. The data in the transmit queue is then transmitted by the UART (in **uart.c**).

Flow control over the radio is maintained by this module:

- When the transmit queue becomes almost full (usually because the UART is not allowed to transmit due to flow control being asserted on the UART), the node indicates this over the radio to its paired node, which stops transmitting further data.
- When the transmit queue begins to empty again, the node indicates this over the radio to its paired node which is then permitted to transmit further data.

When exchanging data between the paired nodes, a single message is used to:

- Indicate the current status of the node (if it is able to transmit and receive, if the receive state is changed, if the node is simply checking the connection).
- Acknowledge data and status changes in the previously received message.
- Send new data to the other device.

In this way, when the nodes are sending data to each other, they will alternate their transmissions, sending both data and acknowledgements in a single message, thus maximising the throughput of data.

The data used by this module is held in the global **WUART_sData** structure defined in **wuart.h**.

This module acts as the hub for the communications between the UART and the radio transceiver, and allocates the queue storage. There are many defines in **wuart.h** to configure the UART port settings, and the sizes and fill levels of the queues.

When **WUART_STATS** is set to **TRUE** in **wuart.h**, statistics on the data being transmitted and received over the radio will be maintained.

When **WUART_LCD** is set to **TRUE** in **wuart.h**, the statistics collected by the various modules will be displayed on the LCD screen of the LCD Expansion Board (DR1215), for monitoring purposes, if available.

3.1.6 uart.c/h

This module provides a high-level interface to the UART hardware, running the UART in an interrupt-driven mode.

The serial queues (allocated in **wuart.c**) are used by the UART code:

- When data is received by the UART, it is added to the receive queue. If the queue becomes almost full, the flow control protocol being used is asserted to prevent the attached device sending any further data. When the queue starts to empty again, the flow control protocol being used is de-asserted allowing further data to be received.
- The UART also monitors the transmit queue, outputting any data in the transmit queue to the attached device. The flow control protocol being used is also monitored - if the attached device indicates that it is not able to receive further data, this condition is detected and no further data is transmitted by the UART until the condition is cleared.

The data used by this module is held in the private **asUart** structure array defined in **uart.h**.

When **UART_STATS** is set to **TRUE** in **uart.h**, statistics on the data being transmitted and received over the UART will be maintained.

3.1.7 queue.c/h

queue.c provides code to manage circular queues.

Each queue is maintained using a **QUEUE_tsData** structure along with a data buffer. These are allocated in **wuart.c** and passed as pointers into **uart.c** during initialisation, so that they can be accessed by both the radio and UART code. The queue functions operate on pointers to a queue and provide the following features:

- Add data to and remove data from a queue.
- Monitor when a queue is almost full and is emptying, to allow flow control to be applied.
- When **QUEUE_STATS** is set to **TRUE** in **queue.h**, statistics on the amount of data and throughput of the queue are calculated.

3.1.8 lcd.c/h

This module provides a high-level interface to display text and numeric values on the LCD screen of the LCD Expansion Board (DR1215), if available.

Code in **wuart.c** gathers the statistics from the other modules in the application and displays them using the functions in **lcd.c**, when the **WUART_LCD** define is set to **TRUE** in **wuart.h**.

3.1.9 rnd.c/h

This module provides a set of random number generating functions, using the hardware random number generator of the JN516x/7x device.

3.1.10 dbg.c/h

This module provides functions allowing debug information to be output to a serial port.

3.2 Functional Overview

3.2.1 Message Format

All messages sent by the application over the radio have the same basic format:

`Start SeqTx SeqRx Command [Data] End`

where:

`Start` is a single character indicating the start of the message, always '!'.

`SeqTx` is a single character ('!' through to '~') that increments with each message transmission, and can be used to detect message received multiple times.

`SeqRx` is a single character ('!' through to '~') that indicates the `SeqTx` in the previous message which this message is responding to. Space is used when the message is not a response.

`Command` is a single character indicating the command being sent. For messages that are commands, these are all upper-case characters, whilst messages that are sent in reply to commands, they are lower-case characters.

`Data` is additional data which may be included depending on the command being sent.

`End` is a single character indicating the end of the message - always 0 (null terminator).

3.2.2 Pairing

Once a node is in the network, it must find another node with which to pair in order to exchange data.

While a node is unpaired, it will regularly broadcast an idle query message to the network. Any other unpaired node that receives such a message will send an idle response message back to the broadcasting node.

Upon receiving an idle response, an unpaired node will exchange a sequence of messages with the other unpaired node in order to form a pair. These messages reflect the steps that a standard dial-up modem follows when negotiating a connection and are illustrated in the Wireless UART State Machine section of this document.

3.2.3 Data Message Format

Once paired together, the paired nodes exchange data using only the data command message. This message may form both a query and reply in a single message. The data command is represented by the command character being 'D'.

The data portion of the message then has the following format:

`Status Acks [DataSeq] [Data]`

where:

`Status` is a single character ('0' through to '?') with the least significant four bits indicating the status of the node:

> `0x01` – Tx enabled, the node is permitted to transmit data over the radio.
>
> `0x02` – Rx enabled, the node is able to receive data over the radio.
>
> `0x04` – Rx changed, the node has changed its Rx enabled state and requires an acknowledgement.

`0x08` – Ping, the node has not sent or received data recently, is checking for the presence of its paired node and requires an acknowledgement.

`Acks` is a single character (‘0’ through to ‘?’) with the least significant four bits indicating acknowledgements of the paired node’s previous message:

`0x01` – Data ack, the node accepted the previously sent packet of data.

`0x02` – Data nak, the node was not able to accept the previously sent packet of data.

`0x04` – Rx change ack, the node is acknowledging the previously sent Rx changed status flag.

`0x08` – Ping ack, the node is acknowledging the previously sent ping status flag.

`DataSeq` is a single character (‘!’ through to ‘~’) that increments with each transmission that includes data. Can be used to ensure data that is resent over the radio is not added to the receiving node’s UART output queue a second time. Only included when data is present in the message.

`Data` is data to be output on the receiving node’s UART. Only included when data is present in the message.

If both nodes need to transmit data at the same time, a node is able to include its next data message with the acknowledgement of the other node's previous message. This results in an efficient use of the radio, as each node alternates its data transmissions.

If a node fails to get a response from its paired node after a number of attempts, it will return to an unpaired state and re-start the process of finding another node to pair with.

3.2.4 Flow Control

End-to-end flow control between two nodes A and B, where A is transmitting to B, works as follows:

1. Node B is instructed by the attached device not to transmit any more data from its UART.
2. Node B's UART transmit queue begins to fill and eventually becomes low on free space.
3. Node B indicates to node A that it is not able to receive further data over the radio.
4. Node A acknowledges that it will not transmit further data to node B.
5. Node B receives the acknowledgement and checks that node A has indicated that it will not transmit further data – if this is not the case then node A sends another status message to node B.
6. Node A is still receiving data on its UART from its attached device.
7. Node A's UART receive queue begins to fill and eventually becomes low on free space.
8. Node A instructs the attached device to stop transmitting data.

Therefore, as the queues begin to fill, the flow control works its way through the system as required. When node B is once again allowed to transmit data to its attached device, the queues will begin to empty and the flags are reversed through the system in a similar way, allowing data to flow once again.

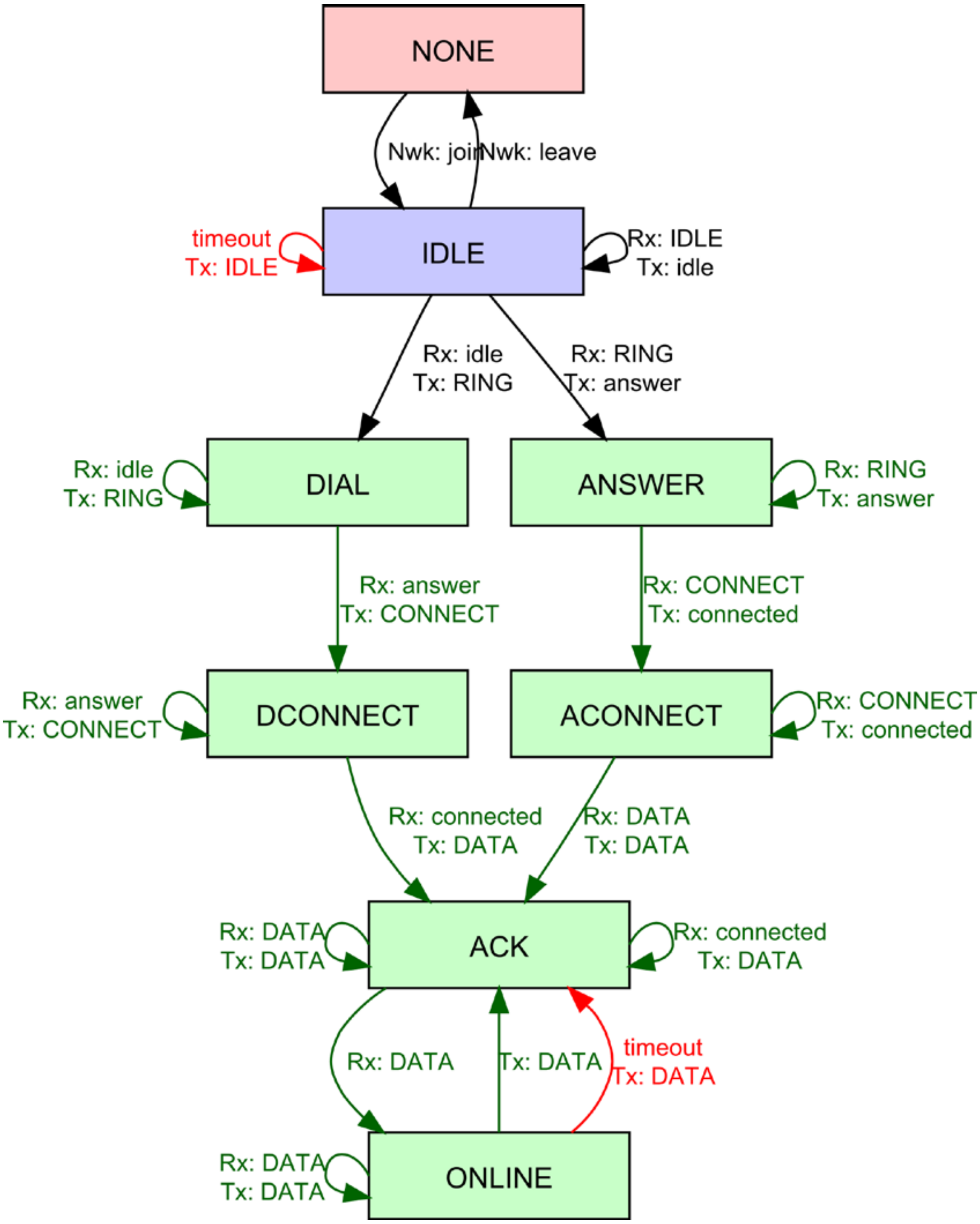
3.2.5 Wireless UART State Machine

The main transitions of the state machine implemented in **wuart.h** are shown in the figure below.

- The black state transition arrows represent transactions caused by network changes or receiving data from any node.
- The green transitions represent data received only from the paired node or the node currently being paired with.
- The red transitions represent transitions caused by the state timing out, either on a regular basis or due to not receiving the expected data.

The following error case transitions are not shown in the figure:

- All the green coloured states timeout to the IDLE state if the expected data is not received. The ACK state only times out to the IDLE state after a number of resends have failed.
- If an unexpected message is received from the paired node in any of the green coloured states, the node returns to the IDLE state responding with an ERROR message.
- If the ERROR message is received from the paired node in any of the green coloured states, the node returns to the IDLE state.
- If messages are received from nodes other than the paired node while in a green coloured state, an ERROR message is returned but the receiving node's state remains unchanged.


NONE
Nwk: join
Nwk: leave
IDLE
timeout
Tx: IDLE
Rx: IDLE
Tx: idle
Rx: idle
Tx: RING
Rx: RING
Tx: answer
DIAL
ANSWER
Rx: idle
Tx: RING
Rx: RING
Tx: answer
Rx: answer
Tx: CONNECT
Rx: CONNECT
Tx: connected
DCONNECT
ACONNECT
Rx: answer
Tx: CONNECT
Rx: CONNECT
Tx: connected
Rx: connected
Tx: DATA
Rx: DATA
Tx: DATA
ACK
Rx: DATA
Tx: DATA
Rx: connected
Tx: DATA
Rx: DATA
Tx: DATA
timeout
Tx: DATA
ONLINE
Rx: DATA
Tx: DATA

4 Compatibility

The software provided with this Application Note is intended to be used with the following NXP hardware and software products:

Product Type	Part Number	Version
Hardware Kit	JN516x-EK001 JN516x-EK004 JN517x-DK005	-
IEEE 802.15.4 SDK	JN-SW-4163 (for JN516x) JN-SW-4263 (for JN517x)	1416 1546
Toolchain	JN-SW-4141 (for JN516x) LPCXpresso (for JN517x)	1308 See SDK Release Notes

For more information on the software requirements for developing with this Application Note, refer to Section 5.

5 Software Requirements for Development

In order to use this Application Note to develop and build your own applications, you need to install the Eclipse-based Integrated Development Environment (IDE) and Software Developer's Kit (SDK) that are appropriate for the chip family which you are using - either JN516x or JN517x:

- **JN516x:** If developing for the JN516x microprocessors, you will need:
  - 'BeyondStudio for NXP' IDE (JN-SW-4141)
  - JN516x IEEE 802.15.4 SDK (JN-SW-4163)

  For installation instructions, refer to the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*.
- **JN517x:** If developing for the JN517x microprocessors, you will need:
  - LPCXpresso IDE
  - JN517x IEEE 802.15.4 SDK (JN-SW-4263)

  For installation instructions, refer to the *JN517x LPCXpresso Installation and User Guide (JN-UG-3109)*.

The LPCXpresso software can be obtained as described in the *JN517x IEEE 802.15.4 SDK Release Notes*, which indicate the version that you will need.

All other resources are available via the **IEEE 802.15.4 for JN516x and JN517x** page of the NXP web site.

Note: The code in this Application Note can be used in either BeyondStudio or LPCXpresso and the process for importing the application into the development workspace is the same for both.

Note: Prebuilt JN5168, JN5169 and JN5179 application binaries are supplied in this Application Note package, but the applications can be rebuilt for other JN516x and JN517x devices (see Section 6).

6 Building and Loading the Application

6.1 Pre-requisites

It is assumed that you have installed the relevant NXP development software on your PC, as detailed in Section 0.

In order to build the application, this Application Note [JN-AN-1069] must be unzipped into the directory:

<IDE installation root>\workspace

where **<IDE Installation root>** is the path in which the IDE was installed. By default, this is:

- **C:\NXP\bstudio_nxp** for BeyondStudio
- **C:\NXP\LPCXpresso_<version>_<build>\lpcxpresso** for LPCXpresso

The **workspace** directory is automatically created when you start the IDE.

All files should then be located in the directory:

…\workspace\JN-AN-1069-IEEE-802.15.4-Serial-Cable-Replacement

There is a sub-directory for each application, each having **Source** and **Build** sub-directories. There will also be sub-directories **JN516x** and **JN517x** containing the project definition files.

6.2 Build Instructions

The software provided with this Application Note can be built for both JN516x and JN517x.

The applications can be built from the command line using the makefiles or from the IDE – makefiles and Eclipse-based project files are supplied.

- To build using makefiles, refer to Section 6.2.1.
- To build using the IDE, refer to Section 6.2.2.

6.2.1 Using Makefiles

This section describes how to use the supplied makefiles to build the applications. Each application (e.g. for Coordinator or End Device) has its own **Build** directory, which contains the makefiles for the application.

The following command line options can be used to configure the built devices:

- `JENNIC_CHIP_FAMILY=JN516x` to build for JN516x microcontrollers
- `JENNIC_CHIP_FAMILY=JN517x` to build for JN517x microcontrollers
- `JENNIC_CHIP=JN5169` to build for a JN5169 microcontroller
- `JENNIC_CHIP=JN5168` to build for a JN5168 microcontroller
- `JENNIC_CHIP=JN5164` to build for a JN5164 microcontroller
- `JENNIC_CHIP=JN5161` to build for a JN5161 microcontroller
- `JENNIC_CHIP=JN5179` to build for a JN5179 microcontroller
- `JENNIC_CHIP=JN5178` to build for a JN5178 microcontroller
- `JENNIC_CHIP=JN5174` to build for a JN5174 microcontroller

To build an application and load it into a JN516x/7x module, follow the instructions below:

1. Ensure that the project directory is located in

   <IDE installation root>\workspace

2. Start an MSYS shell by following the Windows Start menu path:
   All Programs > NXP > MSYS Shell

3. Navigate to the **Build** directory for the application to be built and at the command prompt enter an appropriate `make` command for your chip type, as illustrated below.

   For example, for JN5169:

   ```
   make JENNIC_CHIP_FAMILY=JN516x JENNIC_CHIP=JN5169 clean all
   ```

   For example, for JN5179:

   ```
   make JENNIC_CHIP_FAMILY=JN517x JENNIC_CHIP=JN5179 clean all
   ```

   The binary file will be created in the **Build** directory, the resulting filename indicating the chip type (e.g. **5169**) for which the application was built.

4. Load the resulting binary file into the board. You can do this from the command line using the JN51xx Production Flash Programmer (described in the *JN51xx Production Flash Programmer User Guide (JN-UG-3099)*) – refer to Section 2.1.

6.2.2 Using the IDE (BeyondStudio for NXP or LPCXpresso)

This section describes how to use the IDE to build the demonstration application.

To build the application and load it into JN516x/7x modules, follow the instructions below:

1. Ensure that the project directory is located in

   <IDE installation root>\workspace

2. Start the IDE and import the relevant project as follows:

   a) In the IDE, follow the menu path **File>Import** to display the **Import** dialogue box.

   b) In the dialogue box, expand **General**, select **Existing Projects into Workspace** and click **Next**.

   c) Enable **Select root directory** and browse to the **workspace** directory.

   d) In the **Projects** box, select the project to be imported, only select the project file appropriate for the chip family and IDE you are using and click **Finish**.

3. Build an application. To do this, ensure that the project is highlighted in the left panel of the IDE and use the drop-down list associated with the hammer icon  in the toolbar to select the relevant build configuration – once selected, the application will automatically build. Repeat this to build the other applications.

   The binary files will be created in the relevant **Build** directories for the applications.

4. Load the resulting binary files into the board. You can do this using the integrated Flash programmer, as described in the User Guide for the IDE that you are using.

7 Application Code Sizes

7.1 JN516x Applications

The applications of this Application Note have the following memory footprints on the JN5168 and JN5169 devices, when using the JN516x IEEE 802.15.4 SDK (JN-SW-4163):

Application	Text Size (Bytes)	Data Size (Bytes)	BSS Size (Bytes)
Coordinator_JN5168	38108	288	26787
EndDevice_JN5168	38647	288	27795
Coordinator_JN5169	35792	280	25487
EndDevice_JN5169	36462	284	26503

7.2 JN517x Applications

The applications of this Application Note have the following memory footprints on the JN5179 device, when using the JN517x IEEE 802.15.4 SDK (JN-SW-4263):

Application	Text Size (Bytes)	Data Size (Bytes)	BSS Size (Bytes)
Coordinator_JN5179	38012	664	25519
EndDevice_JN5179	38708	664	26535

Revision History

Version	Notes
1.0	First release
2.0	Re-written for improved performance. JN5148 support added.
3.0	JN5168 and JN5169 support added. JN5148 and JN5139 support dropped.
4.0	JN517x support added.

Important Notice

NXP Semiconductors

For the contact details of your local NXP office or distributor, refer to:

www.nxp.com